APPENDIX – CLAIMS AS AMENDED

1. A method for executing a computer program using a dynamic compiler system, the method comprising:

receiving a byte code of the computer program;

determining if native code is available that corresponds to the byte code;

executing the native code when the native code is available;

when the native code is not available, incrementing a counter;

when the counter is less than a threshold value, interpreting the byte code; and

when the counter is greater than the threshold value, compiling a subroutine containing the byte code.

2. A method as recited in claim 1, wherein the subroutine has a loop structure, wherein the loop structure includes a loop exit test to be performed during each loop iteration, the compiling comprising creating an unrolled loop structure, the unrolled loop structure includes a plurality of loop bodies based on the loop structure.

3. A method as recited in claim 2, wherein the unrolled loop structure includes a loop exit test, the loop exit test is performed once for each iteration of the plurality of loop bodies.

4. A method as recited in claim 2, further comprising building a loop tree based on loops included in the computer program.

5. A method as recited in claim 4, wherein nested loops are represented in the loop tree as child nodes.

6. [Canceled]

7. A method as recited in claim 2, further including performing loop clean up.

8. A method as recited in claim 7, wherein the loop clean-up includes optimizing multiple fall-in loop structures.

9. A method as recited in claim 7, wherein the loop clean-up includes optimizing nested loop structures having invariant operations.

10. A dynamic compiling system, comprising:

an interpreter that interprets byte codes of a computer program, the interpreter tracking a number of times a particular byte code is interpreted and requests that the particular byte code be compiled when the number of times the particular byte code is ~~executed~~ interpreted exceeds a threshold; and

a compiler that compiles the byte codes in response to the requests from the interpreter.

11. A dynamic compiling system as recited in claim 10, wherein the compiler is further capable of creating an unrolled loop structure when compiling an original loop structure of the computer program, the unrolled loop structure including a plurality of loop bodies based on the original loop structure.

12. A dynamic compiling system as recited in claim 11, wherein the unrolled loop structure includes the loop exit test, the loop exit test being performed once for each iteration of the plurality of loop bodies.

13. A dynamic compiling system as recited in claim 11, wherein the compiler is further capable of building a loop tree based on loops included in the computer program.

14. A dynamic compiling system as recited in claim 13, wherein nested loops are represented in the loop tree as child nodes.

15. [Canceled]

16. A dynamic compiler embodied on a computer readable medium, the dynamic compiler comprising:

a code segment that receives a byte code of a computer program;

a code segment that determines whether native code is available that corresponds to a subroutine of the computer program including the byte code, and executes the native code when the native code is available;

a code segment that increments a counter when the native code is not available, the counter tracking a number of times the byte code is interpreted;

a code segment that interprets the byte code when the counter is less than a threshold value; and

a code segment that compiles the subroutine containing the byte code when the counter is greater than the threshold value and native code is not available.

17. The dynamic compiler of claim 16, wherein the subroutine has a loop structure, wherein the loop structure includes a loop exit test to be performed during each loop iteration, the compiling comprising creating an unrolled loop structure, the unrolled loop structure includes a plurality of loop bodies based on the loop structure.

18. The dynamic compiler of claim 17, wherein the code segment that compiles further comprises a code segment that performs loop clean up.

19. The dynamic compiler of claim 18, wherein the loop clean-up includes optimizing multiple fall-in loop structures.

20. The dynamic compiler of claim 19, wherein the loop clean-up includes optimizing nested loop structures having invariant operations.

21. The method of claim 1 wherein the byte code is a backward-branching byte code.

22. The method of claim 1 wherein the byte code is a subroutine call.